Documentación Técnica Tarea Corta #3-Prototipo

Bases De Datos

Profesor: Marco Rivera Meneses

Estudiantes:

Gustavo Adolfo Gamboa Mora 2020023596

María Nicole Valverde Jiménez 2022200481

Isaac Somarribas Montero 20202125516

I Semestre 2024

Descripción de los métodos implementados.

Creación de API/WebService/REST utilizando C# y .NET.

Se desarrolló una aplicación web utilizando Angular.

Se creó una aplicación móvil para los clientes utilizando Android Studio.

Se hizo uso de herramientas de control de versiones, como GitHub, para colaboración y desarrollo en equipo.

Descripción de las estructuras de datos desarrolladas.

Durante el desarrollo de RestTEC, se han utilizado diversas estructuras de datos para almacenar información relevante, como:

Archivos XML con texto en formato JSON para simular la base de datos y así almacenar y gestionar la información de clientes, menú, tipo de platos y usuarios.

De la misma manera se realizan estructuras del lado de la aplicación web para poder almacenar temporalmente las estructuras JSON enviadas por la API.

Descripción detallada de los algoritmos desarrollados.

Se creó una API REST que recibe consultas y envía datos mediante los métodos POST, GET y PUT

El cliente realiza peticiones GET cuando desea obtener alguna información del servidor o bien de la base de datos, por otro lado, realiza peticiones POST cuando desea agregar información a la base de datos. Estas peticiones llaman a los WEBServices los cuales se detallan más adelante.

```
[Route("api/[controller]")]
public class ClientController
{

    private ClientServices _clientServices = new ClientServices();

    [HttpPost("/client")]
    public ClientDto Save([FromBody] ClientDto dto)
    {
        return _clientServices.Save(dto);
    }

    [HttpPut ("/client/id")]
    public ClientDto Update( int id,[FromBody] ClientDto dto)
    {
        return _clientServices.Update(dto, id);
    }

    [HttpGet("/client")]
    public List<ClientDto> GetAll()
    {
        return _clientServices.LoadArchive();
    }
```

WebServices

Se desarolló una clase ClientServices y otra clase MenuServices que contiene las siguientes funciones relacionadas con la gestión de los clientes o del menú:

Save(ClientDto dtoC):

Este metodo recibe un objeto de tipo ClienteDto y lo agrega en una lista interna, luego llama al metodo WriteArchive() para escribir el objeto en un archivo de texto y por ultimo devuelve el objeto ClientDto.

```
public ClientDto Save(ClientDto dtoC)
{
    _clientsDto.Add(dtoC);
    WriteArchive(_clientsDto);
    return dtoC;
}
```

Element(int ssn):

Este método busca un cliente en la lista _clientsDto por su identificadorl (ssn), Recorre la lista y compara el número de seguro social con el proporcionado, si encuentra una coincidencia, devuelve el objeto ClientDto correspondiente; de lo contrario, devuelve null.

```
public ClientDto Element(int ssn)
{
    _clientsDto = LoadArchive();
    int i = 0;
    ClientDto target = null;
    while (i<_clientsDto.Count())
    {
        if (_clientsDto.ElementAt(i).Ssn == ssn)
        {
            target = _clientsDto.ElementAt(i);
            break;
        }
        i++;
    }

    try
    {
        return target;
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        throw;
    }
}
```

Search(int pos):

Este método busca un cliente en la lista _clientsDto por su posición (índice). Recorre la lista y compara la posición con la proporcionada,si encuentra una coincidencia, devuelve el objeto ClientDto correspondiente; de lo contrario, devuelve null.

```
public ClientDto Search(int pos)
{
    _clientsDto = LoadArchive();
    try
    {
        ClientDto target = null;
        foreach (ClientDto dto in _clientsDto)
        {
            if (dto.Ssn == pos)
            {
                return dto;
            }
        }

        return target;
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        throw;
    }
}
```

Age(ClientDto dtoC):

Este método calcula la edad de un cliente utilizando su fecha de nacimiento, resta la fecha de nacimiento de hoy para obtener la diferencia y calcular la edad en años.

```
public int Age(ClientDto dtoC)
{
    DateTime today = DateTime.Today;
    DateTime bDay = dtoC.BirthDate;
    TimeSpan dif = today.Subtract(bDay);
    DateTime firstDay = new DateTime(1, 1, 1);
    int age = (firstDay + dif).Year - 1;
    return age;
}
```

LoadArchive():

Este método carga los datos del archivo "Client.txt" en la lista _clientsDto, Lee todas las líneas del archivo, deserializa cada línea en objetos ClientDto y los agrega a la lista y limpia la lista _clientsDto antes de cargar nuevos datos para evitar duplicados.

```
public List<ClientDto> LoadArchive()
{
    var lines = File.ReadAllLines("./Client.txt");

    _clientsDto.Clear();
    ClientDto dto;
    foreach (var line in lines)
    {
        try
        {
            dto = JsonSerializer.Deserialize<ClientDto>(line);

            _clientsDto.Add(dto);
            Console.WriteLine(dto.Ssn);
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            throw;
        }
    }
    return _clientsDto;
}
```

Se desarolló una clase TiposPlatosServices que contiene las siguientes funciones relacionadas con los tipo de platos:

Search(int id):

Este método busca un tipo de plato en la lista _TiposPlatosServices por su ID, recorre la lista y compara el ID con el proporcionado, si encuentra una coincidencia, devuelve el objeto TiposPlatos correspondiente; de lo contrario, devuelve null.

```
public TiposPlatos Search(int id)
{
    _TiposPlatosServices = LoadArchive();
    try
    {
        TiposPlatos target = null;
        foreach (TiposPlatos pl in _TiposPlatosServices)
        {
            if (pl.IdTipos == id)
            {
                return pl ;
            }
        }

        return target;
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        throw;
    }
}
```

Element(int id):

Este método busca un tipo de plato en la lista _TiposPlatosServices por su ID, recorre la lista y compara el ID con el proporcionado, Si encuentra una coincidencia, devuelve el objeto TiposPlatos correspondiente; de lo contrario, devuelve null.

```
public TiposPlatos Element(int id)
{
    _TiposPlatosServices = LoadArchive();
    int i = 0;
    TiposPlatos target = null;
    while (i<_TiposPlatosServices.Count())
    {
        if (_TiposPlatosServices.ElementAt(i).IdTipos == id)
        {
            target = _TiposPlatosServices.ElementAt(i);
            break;
        }
        i++;
    }

    try
    {
        return target;
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
        throw;
    }
}
```

Problemas encontrados:

- Al realizar la aplicación web se utilizó docker, sin embargo, para algunos compañeros al utilizar esto hacía que la aplicación web no sirviera. Por lo tanto, se

hizo un cambio de esto de manera que al usar la aplicación web por medio de angular no fuera necesario el uso de esta aplicación. La manera final fue más útil y más sencilla de usar. Solamente es necesario usar dos comandos. Además, cada vez que se tenía que usar con el docker, además de volver a cargar el código, también se tenía que eliminar y volver a correr un contenedor. Por lo que hace que el proceso sea más tedioso. Fue una solución util para ahorrar tiempo y espacio de memoria en la computadora. Para la implementación de esto se utilizó el siguiente apoyo. DevExpert –Programacion.(28 de julio del 2022) Create your new website using Angular 17 –part 1 Transforming HTML into Angular 17 Magis. [Video]. Youtube.https://www.youtube.com/watch?v=Lyd9tcttXDA .

- Para la realización de la aplicación móvil se inició usando React, sin embargo, al implementar esto pensábamos que tal vez no era la manera más sencilla de hacerla. Ya que no solo se tenía que hacer el uso de un código que se estaba haciendo en visual studio code y se estaba corriendo la aplicación móvil en un emulador en Android Studio. Además, estábamos teniendo problemas con crear la aplicación y las funciones. Por lo que se procedió a hacer el cambio a usar gradle, esto nos permite que solamente sea necesario usar Android Studio, ya que el código se puede cambiar ahí y se puede volver a cargar de una vez el emulador con los cambios. Además, nos permite observar cómo se puede ver el diseño final de cada pantalla en la parte de layout. Para la implementación de esto se ayudó con el siguiente link:
  DevExpert –Programacion.(28 de julio del 2022) *Curso Android desde cero- Estructura del build gradke de app #9.* [Video]. Youtube. https://www.youtube.com/watch?v=wriDnBRYriY

Evidencias del trabajo en equipo
Bitácora Digital

Gustavo
02-03-2024: Me reuní el grupo para leer el documento y se acordó investigar sobre las herramientas y lenguajes que vamos a necesitar.
03-03-2024: Investigué sobre cómo hacer una API y de cómo usar C# .Net y las qué herramientas usar para hacer la aplicación web.
05-03-2024: Me reuní el grupo para conversar la información previamente investigada, se definieron las herramientas que vamos a usar.

06-03-2024 - 12-03-2024:Trabajé en la elaboración de la aplicación web, utilicé la herramientas PostMan para simular que era la API y realizar pruebas
07-03-2023 - 12-03-2024: Realicé las interfaces para que los usuarios puedan interactuar y creé las funciones necesarias para manipular y mostrar la información obtenida de la base de datos.
14-03-2024 Me reuní con los compañeros para conversar los avances realizados
15-03-2024 - 22-03-2024 Trabajé en el desarrollo de la App Móvil

Nicole
02-03-2024: Me reuní el grupo para leer el documento y se acordó investigar sobre las herramientas y lenguajes que vamos a necesitar.
03-03-2024: Investigué sobre cómo hacer una API y de cómo usar C# .Net y las qué herramientas usar para hacer la aplicación web.
05-03-2024: Me reuní el grupo para conversar la información previamente investigada, se definieron las herramientas que vamos a usar.
06-03-2024:Empecé con la elaboración de la API, hice pruebas de envio y recepción de datos JSON usando los metodos POST y GET con la herramiento PostMan.
08-03-2024 - 12-03-2024: Realicé los servicios relacionados con la manipulación de los clientes y del menú.
14-03-2024 Me reuní con los compañeros para conversar los avances realizados
15-03-2024 - 22-03-2024 Trabajé en el desarrollo de la App Móvil

Isaac
02-03-2024: Me reuní el grupo para leer el documento y se acordó investigar sobre las herramientas y lenguajes que vamos a necesitar.
03-03-2024: Investigué sobre cómo hacer una API y de cómo usar C# .Net y las qué herramientas usar para hacer la aplicación web.
05-03-2024: Me reuní el grupo para conversar la información previamente investigada, se definieron las herramientas que vamos a usar.
07-03-2024: Inicié a trabajar los WebServices, logré leer los datos desde un XML con formato JSON y manipular esa información
10-03-2024: Agregué los servicios para manipular la información relacionada con los tipos de platos.
14-03-2024 Me reuní con los compañeros para conversar los avances realizados
15-03-2024 - 22-03-2024 Trabajé en el desarrollo de la App Móvil

Conclusiones

- Se creó una API REST usando C# .NET que permite conectar la base de datos con los clientes tanto web como en la aplicación móvil.
- Se logró realizar una página web utilizando la herramienta Angular.
- Se logró la comunicación entre la página web y la API utilizando el protocolo HTTP.
- Se desarrolló un prototipo para una aplicación que gestiona el funcionamiento de un restaurante.

Bibliografía

AngularJS (2019-02-05). Recuperado de: https://angular.io/ Bootstrap Themes & Templates (2019-02-05). Recuperado de: https://getbootstrap.com/docs/4.1/getting-started/introduction/

BillWagner. (s/f). .NET documentation. Microsoft.com. Recuperado de https://learn.microsoft.com/en-us/dotnet/

Developer guides. (s/f). Android Developers. Recuperado de https://developer.android.com/guide